Declarative Semantics for Declarative Networking

Jan Van den Bussche
Hasselt University, Belgium

joint work with Tom Ameloot (Hasselt),
Peter Alvaro, Joe Hellerstein, Bill Marczak (Berkeley)

Declarative Networking

UC Berkeley

SIGMOD 2006: Network Datalog [Loo, Hellerstein, et al.]

- use Datalog to program **network protocols**, e.g.:
  - routing (shortest path)
  - peer-to-peer

PODS 2010: Dedalus [Hellerstein et al.]

- use Datalog to program **clusters**:
  - querying distributed databases
  - data-oriented cloud computing

Distributed computing

Hard to program

Two extremes:

- Message-Passing Interface (in C or Fortran)
- SQL-like formalisms (MapReduce, Hive, Pig)

Dedalus offers something in between

Practical language: BLOOM

WebDamLog [Abiteboul et al.]

Distributed transitive closure in Dedalus

Input: binary relation R, distributed among the nodes

Output: transitive closure T of R

$$T(u,v) \mid y \leftarrow R(u,v), \texttt{All}(y).$$
$$T(u,v) \mid y \leftarrow T(u,w), T(w,v), \texttt{All}(y).$$
$$T(u,v) \bullet \leftarrow T(u,v).$$

Two *sending* rules—one *inductive* rule

Distributed emptiness test in Dedalus

Input: nullary relation S, distributed among the nodes

Output: T is true if S is empty, false otherwise

$$\begin{aligned} \mathit{empty}(x) \mid y &\leftarrow \mathtt{Id}(x), \neg S(), \mathtt{All}(y). \\ \mathit{empty}(x) \bullet &\leftarrow \mathit{empty}(x). \\ \mathit{notDone}() &\leftarrow \mathtt{All}(x), \neg \mathit{empty}(x). \\ T() &\leftarrow \neg \mathit{notDone}(). \end{aligned}$$

Last two rules are *deductive*

Distributed database queries

Declarative networking?

Datalog has a nice model-theoretic semantics

Network Datalog only uses Datalog syntax,
lacks a formal semantics

An operational semantics seems most suitable

Dedalus people had crazy idea to use the stable model semantics

We have proven that this actually works!

Operational semantics

Transition system with states containing, for each node x,

- local database (input relations, message relations, memory relations, output relations)
- buffer with messages addressed to x

Transitions: a recipient node is chosen, and a subset of its buffer is delivered

1. apply deductive rules
2. apply inductive rules, sending rules

Datalog with negation

Positive datalog: $T(u,v) \leftarrow R(u,v)$

$$T(u,v) \leftarrow T(u,w), T(w,v)$$

Stratified datalog with negation: $T'(u,v) \leftarrow S(u,v), \neg T(u,v)$

$$T(u,v) \leftarrow R(u,v)$$

$$T(u,v) \leftarrow T(u,w), T(w,v)$$

Unrestricted negation: $Win(x) \leftarrow Move(x,y), \neg Win(y)$

- Well-founded semantics, deterministic
- **Stable model semantics**, nondeterministic

Stable models of $Win(x) \leftarrow Move(x, y), \neg Win(y)$

Given an instance I for $Move$

An expansion M of I to Win is called *stable* if:

1. ground the program on I

2. remove rules that have negative subgoal $\neg Win(a)$ with $Win(a) \in M$

3. remove negative subgoals in rules that remain

4. M should be least fixpoint of resulting positive program

Suppose $I = Move(1, 2), Move(2, 3)$

Ground program: $Win(1) \leftarrow Move(1, 2), \neg Win(2)$
$Win(2) \leftarrow Move(2, 3), \neg Win(3)$

$M = \emptyset$: keep both rules, infer $Win(1)$ and $Win(2)$, not stable

$M = Win(2)$: remove first rule, infer only $Win(2)$, stable

Saccá and Zaniolo's choice construction

$$Other(p,h) \leftarrow Hobby(p,h), Chosen(p,h'), h' \neq h$$
$$Chosen(p,h) \leftarrow Hobby(p,h), \neg Other(p,h).$$

Given relation $Hobby$, a stable model will contain exactly one chosen hobby for each person

Use for sending rule: $T(u,v) \mid y \leftarrow R(u,v), \texttt{All}(y)$

syntactic sugar for $T(y,t,u,v) \leftarrow R(x,s,u,v), \texttt{All}(y),$
$Chosen(x,s,y,u,v,t)$

Every relation gets two extra arguments:
location and timestamp

Deductive, inductive rules

Deductive rule $T() \leftarrow \neg notDone()$

syntactic sugar for $T(x,s) \leftarrow \neg notDone(x,s)$

Inductive rule $T(u,v) \bullet \leftarrow T(u,v)$

syntactic sugar for $T(x,s+1,u,v) \leftarrow T(x,s,u,v)$

Timestamp domain is natural numbers

We obtain a pure Datalog program with negation

Theorem

If the original Dedalus program was negation-free, then the stable models of the resulting Datalog$^{\neg}$ program are exactly the traces of the operational semantics

Theorem

If the original Dedalus program is negation-free, then the **fair** stable models of the resulting Datalog$^\neg$ program are exactly the **fair** traces of the operational semantics

If original program uses negation:

- Deductive rules must be stratified
- Add some extra control rules that express *vector clocks*
- Same theorem obtains

Conclusion

Expressive power: $while$ queries

Study various notions of confluence

References

J.M. Hellerstein. *The declarative imperative—Experiences and conjectures in distributed logic*. PODS 2010 keynote, Indianapolis.

T.J. Ameloot, F. Neven and J. Van den Bussche: *Relational transducers for declarative networking*. PODS 2011, Athens.

P. Alvaro, T.J. Ameloot, J.M. Hellerstein, W. Marczak and J. Van den Bussche: *A declarative semantics for Dedalus.* UC Berkeley EECS Technical Report 2011-120, 2011. Submitted.

T.J. Ameloot and J. Van den Bussche: *On the CRON conjecture*. To appear, Datalog 2.0, Vienna, September 11–13, 2012.

T.J. Ameloot and J. Van den Bussche. *Deciding eventual consistency for a simple class of relational transducers*. ICDT 2012, Berlin.